D-iteration: Evaluation of a Dynamic Partition Strategy

Dohy Hong
Alcatel-Lucent Bell Labs
Route de Villejust
91620 Nozay, France
dohy.hong@alcatel-lucent.com

ABSTRACT

The aim of this paper is to present a first evaluation of a dynamic partition strategy associated to the recently proposed asynchronous distributed computation scheme based on the D-iteration approach. The D-iteration is a fluid diffusion point of view based iteration method to solve numerically linear equations. Using a simple static partition strategy, it has been shown that, when the computation is distributed over K virtual machines (PIDs), the memory size to be handled by each virtual machine decreases linearly with K and the computation speed increases almost linearly with K with a slope becoming closer to one when the number N of linear equations to be solved increases. Here, we want to evaluate how further those results can be improved when a simple dynamic partition strategy is deployed and to show that the dynamic partition strategy allows one to control and equalize the computation load between PIDs without any deep analysis of the matrix or of the underlying graph structure.

Categories and Subject Descriptors

G.1.0 [**Mathematics of Computing**]: Numerical Analysis—*Parallel algorithms*; G.1.3 [**Mathematics of Computing**]: Numerical Analysis—*Numerical Linear Algebra*; C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*

General Terms

Algorithms, Performance

Keywords

Distributed computation, Iteration, Fixed point, Eigenvector.

1. INTRODUCTION

Solving efficiently a very large linear equation systems (and the related initial problems) is a very classical problem and challenge for the algorithm design. The complexity of the problem to solve numerically a very large linear systems may increase rapidly with the dimension of the vector space. There are many known approaches to solve such a class of problems: Gauss elimination, Jacobi iteration, Gauss-Seidel iteration, SOR (successive over-relaxation), Richardson, Krylov, Gradient method, power iteration, QR algorithm etc [11], [31], [4], [10], [25], [33]. And there are more specific approaches in more particular cases when the linear equations are associated to a sparse matrix (in particular, in the context of PageRank equation [26], [6], [7], [3]: power method [30] with adaptation [20] or extrapolation [12], [21], [8], iterative aggregation/disaggregation method [27], [18], [29], adaptive on-line method [2], etc). The case of the symmetric and diagonally dominant (SDD) systems [9], [32], [24] is also a very interesting case that was deeply investigated. In parallel, there have been a lot of researches concerning the distributed computation of the linear equations [5], [19], [28], [23], [22], with a particular interest on asynchronous iteration scheme.

The algorithm proposed here is a new solution for a class of problem we could call diagonally dominant (DD) systems based on the recent research results on the D-iteration. The D-iteration method was initially introduced in [17] to solve numerically the eigenvector of the PageRank equation (the eigenvector defining the score of the page importance). Its applicability in a general linear equation has been described in [16]. The distributed architecture based on this algorithm was first proposed in [13] and then evaluated through simulations in [14] when static partition strategies are applied. It has been shown in [14] that, when the computation is distributed over K virtual machines (PIDs), the memory size to be handled by each virtual machine decreases linearly with K and the computation speed increases almost linearly with K with a slope becoming closer to one when the number N of linear equations to be solved increases. However, those results were obtained under the assumption that the information diffusion cost can be neglected, in particular the computation cost of the fluid quantities to be diffused were neglected. Such an assumption is not realistic when K becomes larger or more precisely when N/K becomes smaller.

Refining and redesigning the algorithms that were proposed in [16, 13, 14, 15], we propose here to revisit the results in [14] and evaluate the benefit of a simple and natural dynamic partition strategy in order to control and equalize the work load of each virtual machine when parallel computation is used. Such a dynamic scheme may be also required when we assume that the underlying graph structure is evolving continuously in time and updates are applied continuously (cf. [15]).

In Section 2, we describe the distributed architecture that is considered in this paper. Section 3 presents the evaluation analysis based on synthetic data and dataset of web graph.

2. DISTRIBUTED ARCHITECTURE

In this paper, we will evaluate the performance of the proposed distributed algorithm focusing to the eigenvector problem associated to PageRank type equation. However,

the algorithm is described here in a more general case. We assume given a square matrix P of size $N \times N$ and an initial condition B (a vector of size N). The D-iteration applied on (P, B) solves X (a vector of size N) satisfying:

$$X = P.X + B.$$

The approach proposed here should work as soon as the spectral radius of P is strictly less than 1 (this is what we could call a diagonally dominant system that was mentioned in the introduction). In particular, the entries of P or B may be positive or negative (cf. [16]). However, for a better intuitive understanding, we chose here to focus on the case where all entries of P are non-negative and implicitly associated to a transition matrix.

2.1 D-iteration: diffusion approach

We recall that the D-iteration is based on the fluid diffusion approach where one step of the iteration consists in choosing a node i_n (n-th step) and diffusing all fluid at node i_n to its children nodes (non zero entries of the i_n-th column of P): at each step of the iteration, we keep two state vectors: the current residual/transient fluids are described by the vector F_n and the history (counting the amount of diffused fluid by each node) of the fluid diffusion by the vector H_n.

Below, an adaptation of the pseudo-code in [14] for the general case:

```
Initialization:
  For i=1..N:
    H[i] := 0;   // History (counter)
    F[i] := B_i; // Fluid

Iteration:
While ( r > Target_Error )
  Choose i;    // node selection
  sent := F[i];
  H[i] += sent;
  F[i] := 0;
  For (j such that p(j,i) != 0):
    F[j] += sent * p(j,i);
  r := |F| = sum_j |F[j]|;
```

When the above scheme converges (DD system), we have asymptotically (when `Target_Error` $\to$ zero) $X = H$.

2.2 Distributed algorithm

We assume that the set $\Omega = \{1, .., N\}$ is partitioned in K sets Ω_k, $k = 1, .., K$ (static or dynamic, see Section 2.5). We set L the number of non zero entries of the matrix P (total number of links).

2.2.1 Local information and diffusion

We distribute the computation tasks of the D-iteration scheme between K virtual machines (we will call PIDs) as follows (cf. [14]):

- each PID_k keeps information on:
  - the set of nodes it is responsible for: Ω_k;
  - the extracted matrix $C_k(P) = (p_{ij})_{i \in \Omega, j \in \Omega_k}$, the column vectors of P corresponding to Ω_k;
  - the marginal fluid vector $[F]_k = ((F)_i)i \in \Omega_k$;
  - the marginal history vector $[H]_k = ((H)_i)i \in \Omega_k$;
  - the previous history vector $[H_{old}]_k = ((H)_i)i \in \Omega_k$, the history vector value at the moment of the last fluid transmission (to other PIDs);
  - its activity state: *active* or *idle* state;
  - the target error value: *target_error*;
- each PID_k maintains two local variables (evaluated periodically):
  - the local residual fluid: $r_k = |[F]_k|$;
  - the fluid to be transmitted: $s_k = |C_k(P)([H]_k - [H_{old}]_k)|$;
- each PID_k applies the local diffusion algorithm (*) below (when not in idle state);
- activity state:
  - initialized to active;
  - PID_k's state is set to idle when

    $$r_k < \max(s_k/10.0, target\_error \times \epsilon/K/10),$$

    where ϵ is a factor depending on P: for PageRank equation, $\epsilon = 1 - damping\_factor$;
- each PID_k select the node to be diffused by a cyclic check-up of elements of Ω_k of the condition:

  $$(F)_i \times w_i > T_k,$$

  where T_k is a threshold value initialized to an arbitrary value larger than $\max_{i \in \Omega_k}(F)_i \times w_i$ and w_i the weight we associate to the node i; the greedy approach would set $w_i = 1$; other candidates are: $w_i = 1/(\#out_i)$ or $w_i = 1/(\#out_i \times \#in_i)$, where $\#out_i$ and $\#in_i$ are respectively the number of of the outgoing links from (number of non zero entries of i-th column of P) and the incoming links to node i (number of non zero entries of i-th line of P). By default, we choose in this paper $w_i = 1/(\#out_i)$. When for all i, the condition $(F)_i \times w_i > T_k$ is not satisfied, we apply: $T_k := T_k/\gamma$ (by default, $\gamma = 1.2$).

```
Local diffusion for PID(k): (*)
  Choose i in Omega_k;
  sent := F[i];
  H[i] += sent;
  F[i] := 0;
  For ( j in Omega_k such that p(j,i) != 0 ):
    F[j] += sent * p(j,i);
```

2.2.2 Fluid exchange

The transmission of fluid from PID_k to other PIDs is done when:

$$s_k \quad > \quad r_k/2. \tag{1}$$

The idea is just to anticipate a bit the moment when s_k and r_k becomes equal. The PIDs ($PID_{k'}$) receiving $received = |[C_k(P)([H]_k - [H_{old}]_k)]_{k'}|$ fluids reinitialize $T_{k'}$ to $\min(T_{k'} \times (r_{k'} + received)/r_{k'}, received)$.

2.3 PID modelling

As in [14], we consider a time stepped approximation for the simulation of the distributed computation cost (for now running on a single PC): during each time step, each PID can execute PID_Speed_k operations. By default, we set: $PID\_Speed_k = PID\_Speed = N/K$ (by default, PIDs are assumed to compute at the same speed).

When a PID is active, it increments `count_active_k` each time an elementary operation (a diffusion from one node to another node in the same Ω_k set, which roughly corresponds to a product of one entry $(F)_j$ with one entry of the matrix $(P)_{ij}$ and the addition of the product to $(F)_i$) is done.

Every time step, we set a local counter that counts the number of elementary operations that are not consumed (because entering in the idle state). When a PID is idle, the *wasted* operations are then added to `count_idle_k`.

In the following, the number of iterations is defined as the normalized quantity:

$$\frac{count\_active\_k + count\_idle\_k}{L}$$

so that it can be easily compared to the cost of one matrix-vector product, or one iteration in power iteration.

2.4 Computation cost

The computation effort of PID_k is indirectly estimated through `count_active_k`. This counter is incremented:

- by one each time there is a local diffusion from one node to another;
- by one to the receiver for each diffusion to one node (during fluid exchange) managed by the receiver; for the sender, we increment by one for each diffusion coming from $C_k(P)([H]_k - [H_{old}]_k)$: this is the quantity that was underestimated in [14];
- by the number of nodes re-affected for the partition set adaptation.

2.5 Partition sets

2.5.1 Static partition sets

As in [14], we consider two simple K partition sets for comparison purpose:

- Uniform partition: $\Omega_1 = \{1, 2, ..., N/K\}$, $\Omega_2 = \{N/K + 1, 2, ..., 2 \times N/K\}$, etc
- Cost Balanced (CB) partition: $\Omega_k = \{\omega_k, \omega_k+1, ..., \omega_{k+1}-1\}$ such that $\sum_{n=\omega_k}^{\omega_{k+1}-1}(\#out_n) = L/K$,

such that $\{1, ..., N\} = \Omega = \cup_k \Omega_k$. The intuition of the cost balanced partition is that when we apply the diffusion iteration on all nodes of each Ω_k, the diffusion cost is constant. The main reason why we chose this is the simplicity of its computation [14].

2.5.2 Dynamic partition sets

In the initial state, we start with the uniform or CB partition sets. Then, we update the following quantity every time step (PID_Speed operations in active or idle state):

$$slope\_k := slope\_k \times (1-\eta) - log(r_k + s_k + \varepsilon)/log(10.0) \times \eta$$

where $\varepsilon = target\_error/K/1000$ is added to avoid undefined value of $slope\_k$. The quantity $-slope\_k$ measures the moving averaged value of the exponent (base 10) of $r_k + s_k$: if we plot the curve $r_k + s_k$ as a function of the number of iteration (normalized) in logscale on y-axis, the exponent represents the slope of the curve. By default, we used $\eta = 0.5$.

Then, every time step, we compute k which maximize and minimize $slope\_k$ (resp. $i_{\max}$, $i_{\min}$). If the difference is more than 50%:

$$\text{if } (slope\_min < slope\_max + \log(0.5)/\log(10.0))$$

then, we reaffect:

$$|\Omega_{i_{\min}}| \times \min\left(\frac{slope\_min + 1}{slope\_max + 1}, 0.1\right)$$

nodes from $\Omega_{i_{\min}}$ to $\Omega_{i_{\max}}$ ($i_{\min}$ identifies the slowest PID).

To minimize the oscillation behaviour, the sets that are just re-affected (decreased or increased) can not be re-affected during the next Z steps (by default $Z = 10$).

When the Ω_k set is re-affected, we increment its operation cost counter `count_active_k` (by the number of nodes modified for $\Omega_{i_{\min}}$ and $\Omega_{i_{\max}}$).

3. EXPERIMENTS AND EVALUATION

3.1 Synthetic data

We first used a synthetic data generated as follows: assuming a power-law $1/k^\alpha$ ($\alpha = 1.5$ used here) for the in-degree and the out-degree distribution, we generated random links between pair of nodes (see [17] for more details).

3.1.1 Analysis of $K = 2$: $N = 1000$

Let us start with 2 PIDs case for an easier illustration of the problem. Figure 1 shows the plots of the convergence speed (given by the ratio of $r_k + s_k$ and the number of iterations) in logscale on y-axis, when starting with a static partition sets of $250 + 750$, $500 + 500$ and $750 + 250$ (in this case, $L = 9543$).

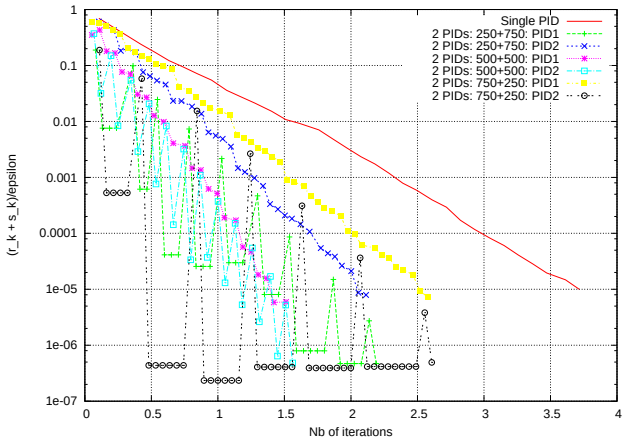

Figure 1: Illustration of convergence speed: fluid exchange cost neglected.

When the Ω_k is not set correctly (too big or too small), the gain of the parallelism is reduced. We remark that when the fluid exchange cost is neglected, $K = 2$ ($500 + 500$ case)

can improve by factor above 2. Figure 2 shows the plots of the convergence speed integrating the fluid exchange cost: the relative gain to the single PID case is much less important than previous results, illustrating the importance of this factor even when K is small.

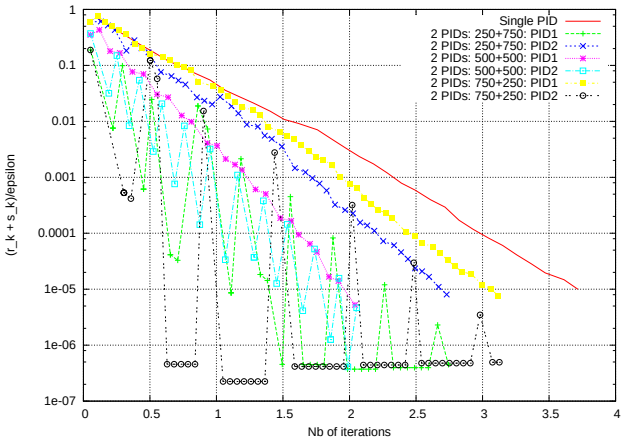

Figure 2: Illustration of convergence speed: fluid exchange cost integrated.

Figure 3 illustrates the impact of the partition adaptation on the convergence speeds that are made closer: in this case, we took initial partition sets of 750-250 and let the system self adapts.

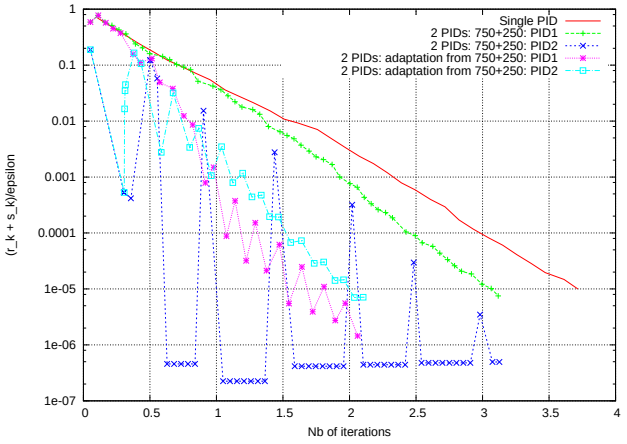

Figure 3: Illustration of the impact of the dynamic partition.

Figure 4 illustrates the evolution of the partition sets when starting from 750-250 (here, we took $Z = 1$ for a quicker adaptation).

Table 1 gives a comparative computation time (number of iterations of the slowest PID) of different approaches for $K = 1$ to 128 ($Z = 10$): by construction, this can be considered as the most favourable situation for the uniform partition (links are independently and identically distributed to all nodes). We see that the dynamic strategy can still improve in almost all situations (but not too much).

To further illustrate the advantage of the dynamic adaptation, we biased the nodes ordering replacing the complete random one (previous) by the number of outgoing links (cf.

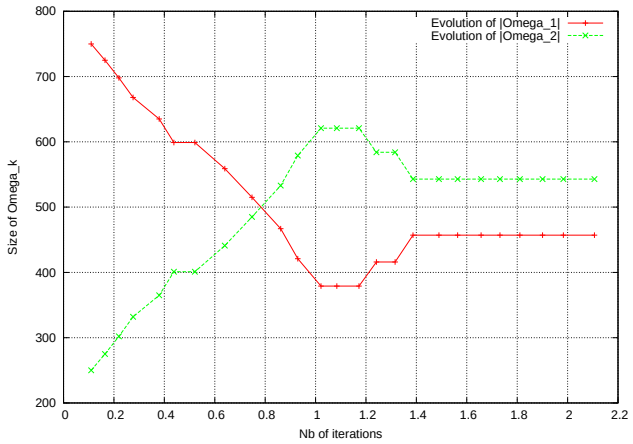

Figure 4: Illustration of the evolution of the dynamic partition.

	From Unif. partition		From CB partition	
K	Static	Dynamic	Static	Dynamic
1	2.39	2.39	2.39	2.39
2	1.39	1.25	1.31	1.38
4	0.85	0.85	0.81	0.80
8	0.56	0.53	0.49	0.47
16	0.37	0.35	0.43	0.38
32	0.29	0.27	0.31	0.26
64	0.26	0.22	0.30	0.24
128	0.26	0.26	0.35	0.29

Table 1: Illustration of the computation time for a target error of $1/N$: $N = 1000$.

	From Unif. partition		From CB partition	
K	Static	Dynamic	Static	Dynamic
1	3.79	3.79	3.79	3.79
2	3.07	2.96	2.83	2.33
4	2.48	2.16	3.42	2.68
8	1.97	1.53	5.09	2.63
16	1.57	1.02	6.01	2.40

Table 2: Illustration of the computation time for a target error of $1/N$: $N = 1000$. Nodes are ordered by the number of outgoing links.

	From Unif. partition		From CB partition	
K	Static	Dynamic	Static	Dynamic
1	4.96	4.96	4.96	4.96
2	3.65	3.48	3.55	3.02
4	2.97	2.03	2.57	1.91
8	2.93	1.69	2.48	1.62
16	3.14	1.35	2.28	1.25

Table 3: Illustration of the computation time for a target error of $1/N$: $N = 1000$. Nodes are ordered by the number of incoming links.

Table 2): we see that the CB static strategy is not good and when $K \geq 4$ its performance is even degraded.

The results of the case when the nodes are ordered by the number of incoming links are shown in Table 3: here the uniform partition is the worst one.

Globally, what we observe is that when N/K becomes too small, the gain is limited or the performance may be even degraded due to the fluid exchange cost. Finally, we observe a very good stability/performance of the dynamic partition strategy in all situations.

3.2 Web graph datasets

For the evaluation purpose, we experimented the dynamic partition strategy on a web graph imported from the dataset `uk-2007-05@1000000` (available on [1]) which has 41,247,159 links on 1,000,000 nodes (45,766 dangling nodes).

Below we vary N from 1,000 to 100,000 extracting from the dataset the information on the first N nodes.

Figure 5 shows the summarized results on the convergence speeds (normalized to the convergence cost for $K = 1$) for $N = 1000, 10000, 100000$ starting from the uniform partition (unfortunately, we could not yet handle $N = 1000000$ case because of the memory limitation on a single PC). We clearly see that because of the fluid exchange cost, the convergence becomes slower when K is too large compared to N and that for larger N the optimal K value is larger.

Figure 6 shows the summarized results on the convergence speeds starting from the CB partition. Figure 5 and Figure

N	L (nb links)	L/N	D (Nb dangling nodes)
1000	12,935	12.9	41 (4.1%)
10000	125,439	12.5	80 (0.8%)
100000	3,141,476	31.4	2729 (2.7%)

Table 4: Extracted graph: $N = 1000$ to 100000.

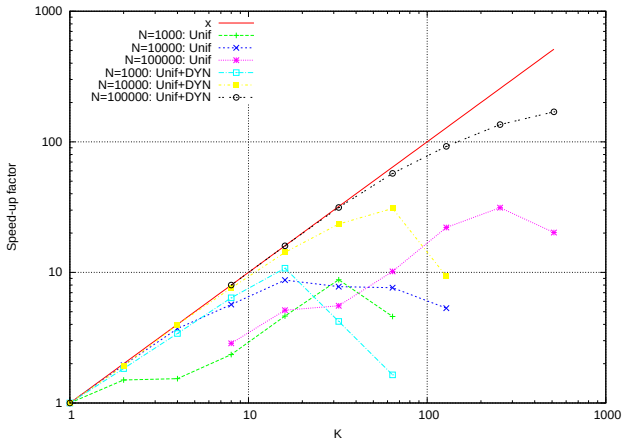

Figure 5: Convergence speed-up factor: starting from uniform partition.

6 correct results reported in [14] (fluid exchange cost was underestimated) when N/K is small. However, the main conjecture/result which states that, when the computation is distributed over K virtual machines, the computation speed increases almost linearly with K with a slope becoming closer to one when the number N of linear equations to be solved increases is still true: we conjecture that the slope goes to one asymptotically for large N/K and this is very clearly visible in the curves of the dynamic partition based approaches (Unif+DYN or CB+DYN) when N is increased.

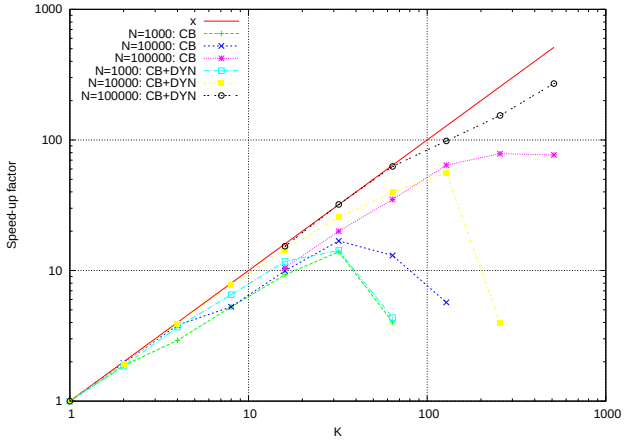

Figure 6: Convergence speed-up factor: starting from CB partition.

Figure 7 shows the consequence on the proportion of the PIDs' idle state

$$\frac{\sum_k count\_idle\_k}{\sum_k (count\_active\_k + count\_idle\_k)}$$

when different approaches are applied ($N = 10000$). We see a clear reduction of the idle state with the dynamic strategy when the fluid exchange is not dominant.

Figure 8 shows the typical result of two different convergence speeds: in this case PID2 is the slowest one. The fluid exchange is done every about 1.2 iterations which is clearly visible here. We can see that PID1 can enter in the

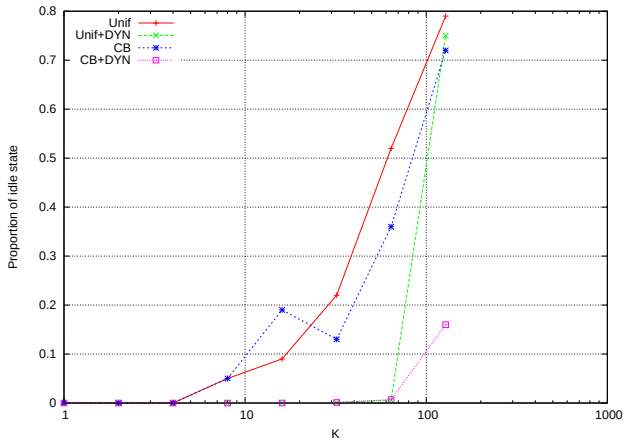

Figure 7: Proportion of the idle state: $N = 10000$**.**

idle state because it is waiting for inputs from PID2 (for x between 4 and 5, between 6.5 and 7.5 etc) when it reaches the target value $\max(s_k/10.0, target\_error \times \epsilon/K/10)$: this is globally not optimal in terms of the PID's computation capacity utilization.

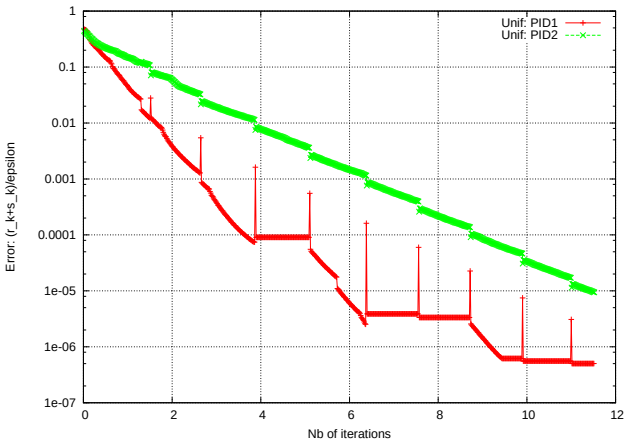

Figure 8: Evolution of convergence: $K = 2$, $N = 100000$ **with static uniform partition.**

Figure 9 shows the impact of the dynamic partition starting from the uniform partition for the same case than Figure 8. The corresponding evolution of the partition sets is shown in Figure 10.

Figure 11 shows the evolution of the convergence of PIDs with static CB partition: because CB is based on a heuristic simplification, it does not guarantee the same computation effort for the two PIDs.

Figure 12 shows the evolution of the convergence with $K = 4$ with the static uniform partition, the static CB partition and the dynamic partition starting from the uniform and CB partitions: in this case, the benefit of the dynamic partition is very clear with an acceleration by a factor above 3.

Figure 13 shows the evolution of the convergence with $K = 8$ with the static uniform partition, the static CB partition and the dynamic partition starting from the uniform partition: in this case, the speed-up factor with the dynamic

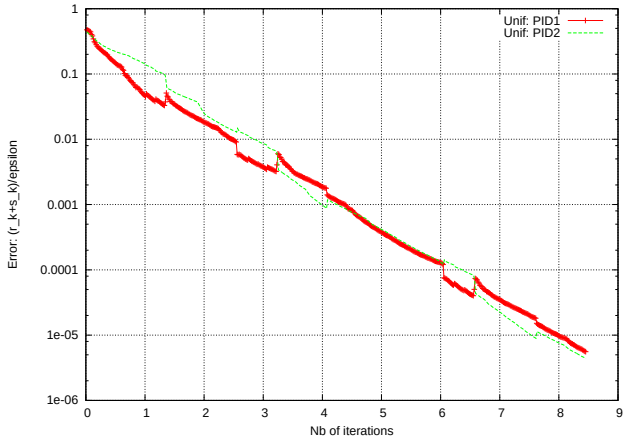

Figure 9: Evolution of convergence: $K = 2$, $N = 100000$**. Dynamic partition from the uniform partition.**

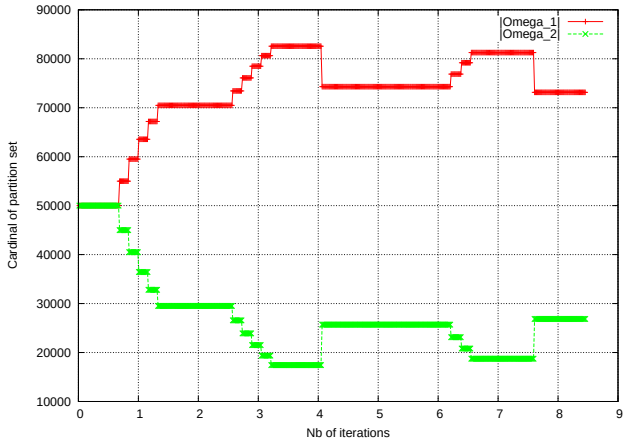

Figure 10: Evolution of partition sets: $K = 2$, $N = 100000$**.**

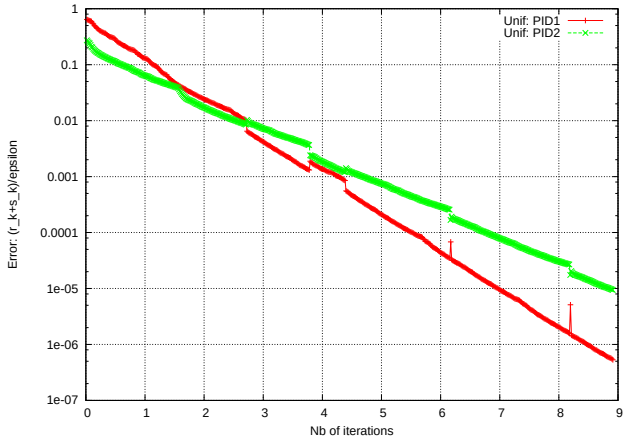

Figure 11: Evolution of convergence: $K = 2$, $N = 100000$ **with static CB partition.**

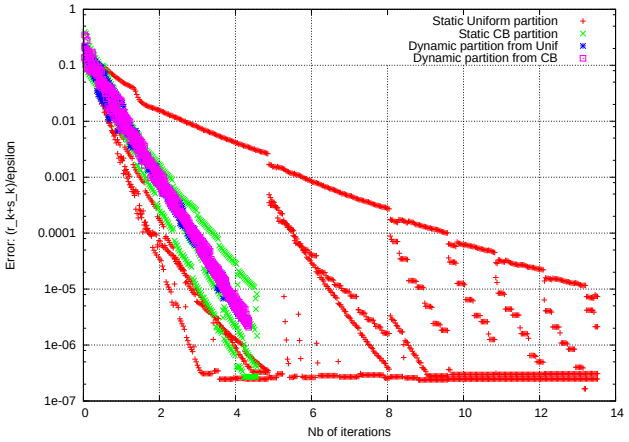

Figure 12: Evolution of convergence: $K = 4$, $N = 100000$. Comparison of static unif., static CB and dynamic from unif. and CB.

strategy is above 2.

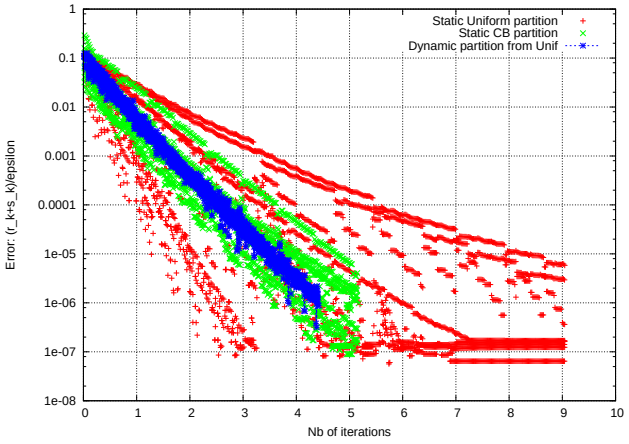

Figure 13: Evolution of convergence: $K = 8$, $N = 100000$. Comparison of static unif., static CB and dynamic from unif.

Figure 14 shows the result of 128 different convergence speeds with $K = 128$: in this case, we can identify 2 slowest PIDs. The computation capacities of 126 other PIDs are likely to be wasted.

Figure 15 shows the impact of the dynamic partition starting from the uniform partition for the same case than Figure 14. In this case, the speed-up factor is about 4 thanks to a better computation effort redistribution between PIDs.

Figure 16 and Figure 17 show the global convergence (an upper bound on the L_1 norm to the distance) for different approaches ($N = 10000$): the benefit of the dynamic adaptation is more visible for $K \geq 8$. Note that those curves must be strictly decreasing function: we observe here some local fluctuation due to the artefact of the time stepped approximation (linked to the fluid exchange cost): when the fluid exchange cost becomes important, the concerned PID is likely to be frozen during that time.

Figure 18 and Figure 19 show the global convergence for $N = 100000$: when K and N are larger, the analysis be-

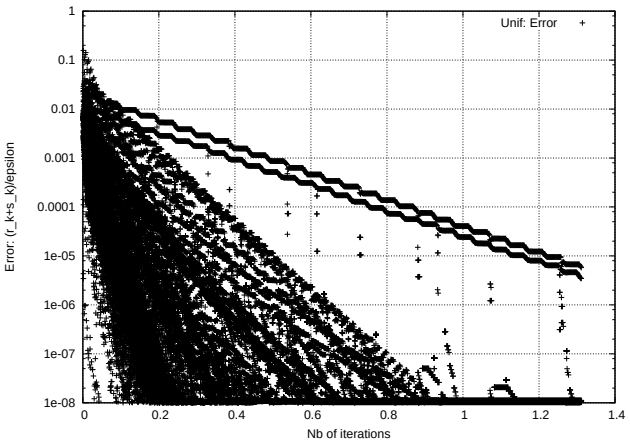

Figure 14: Evolution of convergence: $K = 128$, $N = 100000$ with static uniform partition.

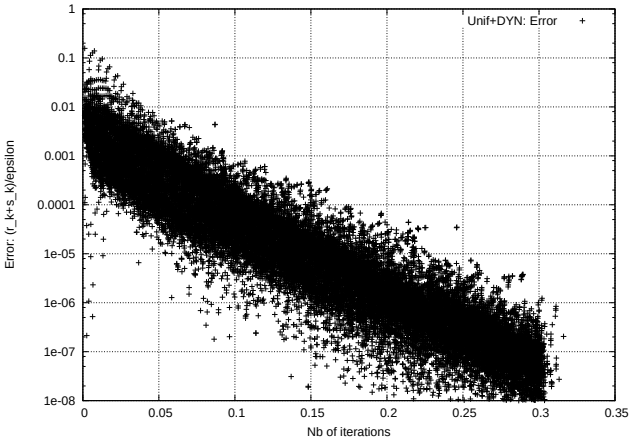

Figure 15: Evolution of convergence: $K = 128$, $N = 100000$. Dynamic partition from the uniform partition.

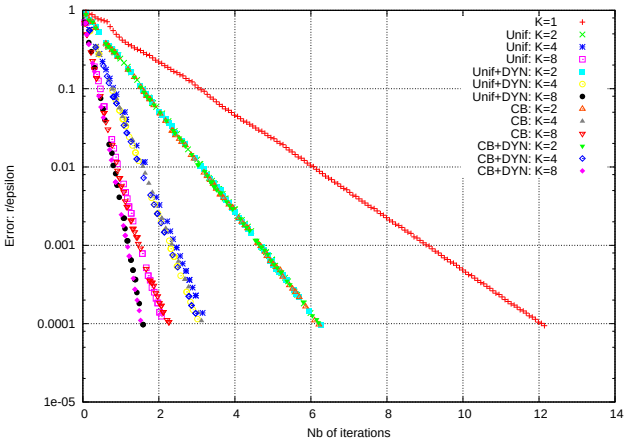

Figure 16: Global convergence: $N = 10000$. For $K = 2, 4, 8$.

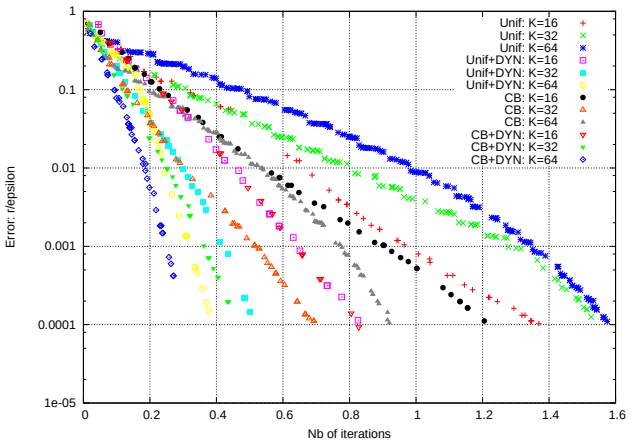

Figure 17: Global convergence: $N = 10000$**. For** $K = 16, 32, 64$**.**

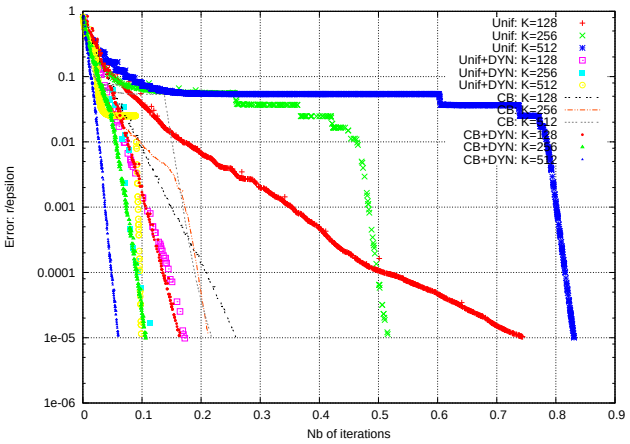

Figure 19: Global convergence: $N = 100000$**. For** $K = 128, 256, 512$**.**

comes much more complex: we can observe significant and sudden slope modification during the iteration. See for instance Unif+DYN or CB curves for $K = 512$ in Figure 19. One of very visible effect is the impact of the fluid exchange cost which is increased for larger value of K.

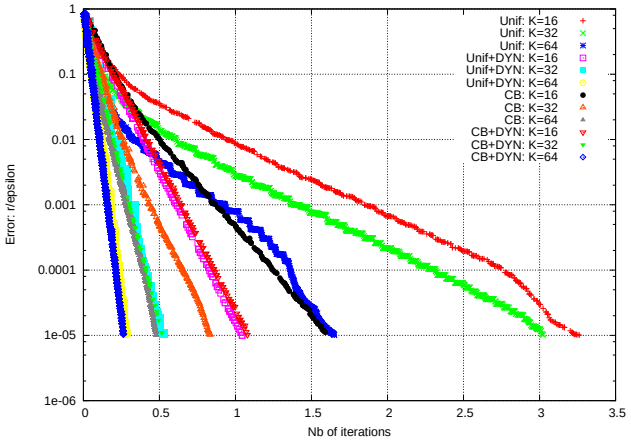

Figure 18: Global convergence: $N = 100000$**. For** $K = 16, 32, 64$**.**

The above results show in particular (and this is not surprising) that it is not necessarily better to increase K and an optimal K need to be applied (for a given vector size N). This may suggest the possibility of considering a further adaptive scheme where we could also dynamically adjust the number of PIDs: we hope to address this issue in a future work. What we propose here is a first simple candidate to highlight the potential of the approach. From this first step, one may explore a lot of variants (for instance, we should favour partition sets such that there are more links inside the Ω_k sets; we could also define the number of nodes to be re-affected, when modification required, based on its CB evaluation, etc).

4. CONCLUSION

In this paper, we presented an adaptive dynamic partition strategy applied to a distributed computation architecture of the D-iteration method. Through experiments on synthetic data and real dataset, we showed that a dynamic partition strategy brings a robustness and a better efficiency guarantee compared to the static partition strategy, especially when N is large. We believe that, even though this is preliminary results that need to be confirmed by a real deployment of a distributed system with possibly further adaptation/modification of the algorithm design, we showed here the potential of a new promising distributed computation architecture to solve a very large diagonal dominant class of linear systems.

5. REFERENCES

[1] http://law.dsi.unimi.it/datasets.php.

[2] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. *WWW2003*, pages 280–290, 2003.

[3] A. Arasu, J. Novak, J. Tomlin, and J. Tomlin. Pagerank computation and the structure of the web: Experiments and algorithms, 2002.

[4] R. Bagnara. A unified proof for the convergence of jacobi and gauss-seidel methods. *SIAM Review*, 37, 1995.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: numerical methods.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[6] M. Bianchini, M. Gori, and F. Scarselli. Inside pagerank. *ACM Trans. Internet Techn.*, 2005.

[7] P. Boldi, M. Santini, and S. Vigna. Pagerank: Functional dependencies. *ACM Trans. Inf. Syst.*, 27:19:1–19:23, November 2009.

[8] C. Brezinski, M. Redivo-Zaglia, and S. Serra-Capizzano. Extrapolation methods for pagerank computations. *Comptes Rendus Acad. Sci.*, pages 393–397, 2005.

[9] J. R. Bunch, J. W. Demmel, and C. F. van Loan. The strong stability of algorithms for solving symmetric linear systems. *SIAM J. Matrix Anal. Appl.*, 10:494–499, October 1989.

[10] J. G. F. Francis. The QR transformation, I. *The Computer Journal*, 4(3):265–271, 1961.

[11] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[12] T. H. Haveliwala, S. D. Kamvar, D. Klein, C. D. Manning, and G. H. Golub. Computing pagerank using power extrapolation. *Technical report, Stanford University*, 2003.

[13] D. Hong. D-iteration based asynchronous distributed computation. *arXiv, http://arxiv.org/abs/1202.3108*, February 2012.

[14] D. Hong. D-iteration: Evaluation of the asynchronous distributed computation. *submitted, http://arxiv.org/abs/1202.6168*, February 2012.

[15] D. Hong. D-iteration: Evaluation of the update algorithm. *arXiv, http://arxiv.org/abs/1202.6136*, February 2012.

[16] D. Hong. D-iteration method or how to improve gauss-seidel method. *arXiv, http://arxiv.org/abs/1202.1163*, February 2012.

[17] D. Hong. Optimized on-line computation of pagerank algorithm. *submitted, http://arxiv.org/abs/1202.6158*, 2012.

[18] I. C. F. Ipsen and S. Kirkland. Convergence analysis of a pagerank updating algorithm by langville and meyer. *SIAM J. Matrix Anal. Appl.*, 27(4):952–967, 2006.

[19] M. Jelasity, G. Canright, and K. Engø-Monsen. Asynchronous distributed power iteration with gossip-based normalization. In *Euro-Par 2007 Parallel Processing*, volume 4641 of *Lecture Notes in Computer Science*, pages 514–525. 2007.

[20] S. D. Kamvar, T. H. Haveliwala, and G. H. Golub. Adaptive methods for the computation of pagerank. *Linear Algebra Appl.*, pages 51–65, 2004.

[21] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Extrapolation methods for accelerating pagerank computations. *Proc. of International World Wide Web Conference (WWW2003)*, pages 261–270, 2003.

[22] C. Kohlschütter, P.-A. Chirita, R. Chirita, and W. Nejdl. Efficient parallel computation of pagerank. In *In Proc. of the 28th European Conference on Information Retrieval*, pages 241–252, 2006.

[23] G. Kollias, E. Gallopoulos, and D. B. Szyld. Asynchronous iterative computations with web information retrieval structures: The pagerank case. *CoRR*, abs/cs/0606047, 2006.

[24] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving sdd linear systems, 2010.

[25] V. N. Kublanovskaya. On some algorithms for the solution of the complete eigenvalue problem. *USSR Computational Mathematics and Mathematical Physics*, 3:637–657, 1961.

[26] A. N. Langville and C. D. Meyer. Deeper inside pagerank. *Internet Mathematics*, 1(3), 2004.

[27] A. N. Langville and C. D. Meyer. Updating the stationary vector of an irreducible markov chain with an eye on google's pagerank. *Technical report, North Carolina State University*, 2004.

[28] B. Lubachevsky and D. Mitra. A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit spectral radius. *J. ACM*, 33(1):130–150, Jan. 1986.

[29] I. Marek and P. Mayer. Convergence theory of some classes of iterative aggregation/disaggregation methods for computing stationary probability vectors of stochastic matrices. *Linear Algebra Appl.*, pages 177–200, 2003.

[30] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Technical Report Stanford University*, 1998.

[31] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[32] D. A. Spielman and S.-H. Teng. Solving sparse, symmetric, diagonally-dominant linear systems in time $o(m^{1.31})$. *CoRR*, cs.DS/0310036, 2003.

[33] R. von Mises and H. Pollaczek-Geiringer. Praktische verfahren der Gleichungsauflösung. *ZAMM - Zeitschrift für Angewandte Mathematik und Mechanik*, 9:152–164, 1929.